华中科技大学

课 程 实 验 报 告

课程名称：C++程序设计

实验名称：面向对象的矩阵运算编程

院　　系 ：计算机科学与技术

专业班级 ：　CS2006

学　　号 ：　U202015471

姓　　名 ：　杨释钧

指导教师 ：　纪俊文

2021 年 12 月 15 日

目录

1 需求分析

1.1 题目要求

矩阵 MAT 是行列定长的二维数组。常见的矩阵运算包括矩阵的加、减、乘、转置和赋值等运算。请对矩阵 MAT 类中的所有函数成员编程，并对随后给出的 main( )函数进行扩展，以便完成矩阵及其重载的所有运算符的测试。输出矩阵元素时整数用”%6ld” 或”%6lld”打印，浮点数用”%8f”或”%8lf”打印，最后一行用换行符结束”\n”。至少要测试两种实例类 MAT<int>和 MAT<long long>。

```
#define _CRT_SECURE_NO_WARNINGS
#include <iomanip>
#include <exception>
#include <typeinfo>
#include <string.h>
using namespace std;
template <typename T>
class MAT {
    T* const e;                                  //指向所有整型矩阵元素的指针
    const int r, c;                              //矩阵的行 r 和列 c 大小
public:
    MAT(int r, int c);                           //矩阵定义
    MAT(const MAT& a);                  //深拷贝构造
    MAT(MAT&& a)noexcept;               //移动构造
    virtual ~MAT()noexcept;
    virtual T* const operator[ ](int r);//取矩阵 r 行的第一个元素地址，r 越界抛异常
    virtual MAT operator+(const MAT& a)const;      //矩阵加法，不能加抛异常
    virtual MAT operator-(const MAT& a)const; //矩阵减法，不能减抛异常
    virtual MAT operator*(const MAT& a)const;      //矩阵乘法，不能乘抛异常
    virtual MAT operator~()const;                        //矩阵转置
    virtual MAT& operator=(const MAT& a);          //深拷贝赋值运算
    virtual MAT& operator=(MAT&& a)noexcept;    //移动赋值运算
    virtual MAT& operator+=(const MAT& a);         //“+=”运算
    virtual MAT& operator-=(const MAT& a);         //“-=”运算
    virtual MAT& operator*=(const MAT& a);         //“*=”运算
   //print 输出至 s 并返回 s：列用空格隔开，行用回车结束
    virtual char* print(char* s)const noexcept;
};
```

```
int main(int argc, char* argv[ ])                    //请扩展 main()测试其他运算
{
    MAT<int>     a(1, 2), b(2, 2), c(1, 2);
      char t[2048];
      a[0][0] = 1;              //类似地初始化矩阵的所有元素
      a[0][1] = 2;              //等价于“*(a.operator[ ](0)+1)=2;”即等价于“*(a[0]+1)=2;”
      a.print(t);           //初始化矩阵后输出该矩阵
      b[0][0] = 3;    b[0][1] = 4;        //调用 T* const operator[ ](int r)初始化数组元素
      b[1][0] = 5;    b[1][1] = 6;
      b.print(t);
      c = a * b;                          //测试矩阵乘法运算
      c.print(t);
      (a + c).print(t);                   //测试矩阵加法运算
      c = c - a;                          //测试矩阵减法运算
      c.print(t);
      c += a;                                 //测试矩阵“+=”运算
      c.print(t);
      c = ~a;                                 //测试矩阵转置运算
      c.print(t);
      return 0;
}
```

1.2 需求分析

矩阵的数据结构可以用一个二维数组来表示，由于 new 运算符在给数组分配空间时，表示大小最多只允许使用一个变量，由于题设所要求的矩阵行列均不固定，所以不妨使用一维数组 matrix[r * c]来代替 matrix[r][c]。

在实现了题目基本要求的矩阵基本运算之后可以选择增加矩阵的其他运算，比如给定某个矩阵，求其逆矩阵，或者已知一个矩阵，求其传递闭包，而这个运算也正是实验五所要用到的运算。

2 系统设计

2.1 概要设计

本矩阵类中使用一维数组模拟二维数组(一般矩阵)，通过行数 r 和列数 c 的简单运算实现对矩阵特定行列的数据成员的访问。

类的实例数据成员包括：一个模板类型的一维数组，以及表示矩阵行数和列数的整型变量 r 和 c；类的函数成员，从功能分类的角度可以分为初始化、赋值、运算三部分，如图 2.1 所示。

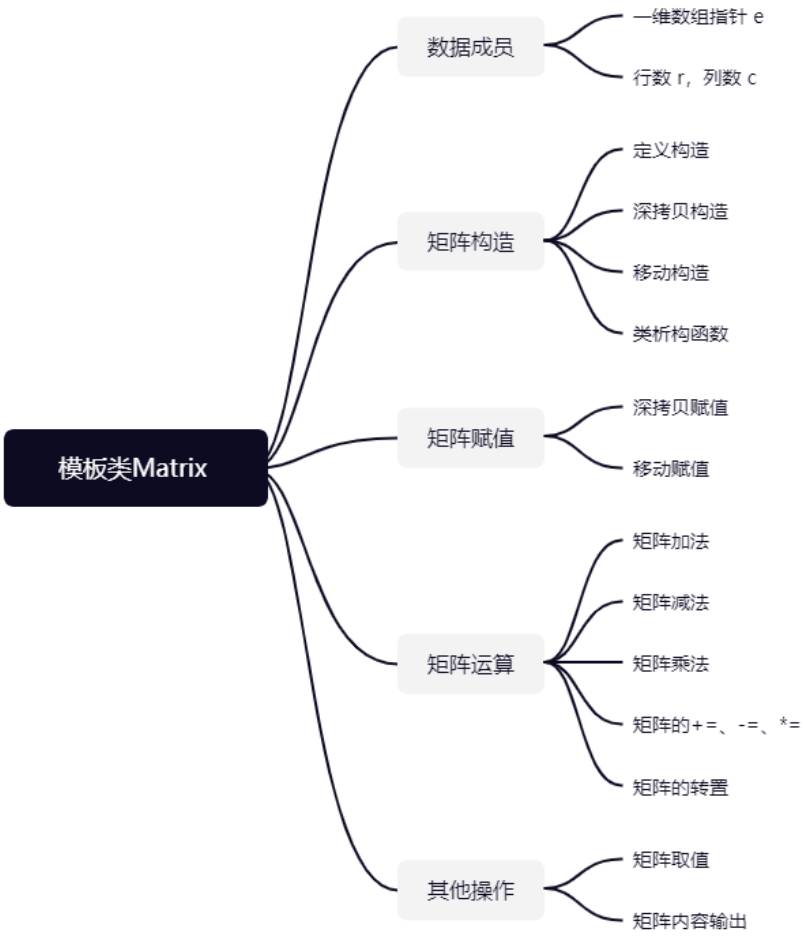

图 2-1 matrix 类的结构

2.2 详细设计

1. 定义构造函数 MAT(int r, int c)；

   仅需给对象的行数列数分别赋值 r 和 c 即可。

2. 深拷贝构造函数 MAT(const MAT& a)；

使用 new 运算符给构造对象的数组指针 e 分配与 a 中数组等大的空间，将构造对象行数列数分别赋值为 a.r 和 a.c，将 a 中数组内容赋值给构造对象数组中对应位置。

3. 移动构造函数 MAT(MAT&& a)；

将构造对象的数组指针、行数和列数分别赋值为 a 中数组的首地址、a.r、a.c；然后将 a 中数组指针、r、c 均赋值为 0 表示已被移动。

4. 析构函数~MAT()；

使用 delete 运算符释放数组 e 的占用空间，并将指针 e 赋值为 nullptr。

5. 取行地址 T* const MAT<T>::operator[ ](int r)；

如果输入的参数 r 越界则抛出异常；否则返回 e[r * c]的地址。

6. 矩阵加法 MAT<T> MAT<T>::operator+(const MAT& a)；

如果参与加法的两个矩阵有行数或者列数不相等的情况，抛出异常；否则将数组中对应成员的和放入一个新构造的 matrix 类对象的数组的对应位置中，函数结束时返回该新对象。

7. 矩阵减法 MAT<T> MAT<T>::operator-(const MAT& a)；

如果参与减法的两个矩阵有行数或者列数不相等的情况，抛出异常；否则将数组中对应成员的差放入一个新构造的 matrix 类对象的数组的对应位置中，函数结束时返回该新对象。

8. 矩阵乘法 MAT<T> MAT<T>::operator*(const MAT& a)；

如果参与减法的两个矩阵有行列数不匹配的情况，抛出异常；否则将数组中对应成员的乘积放入一个新构造的 matrix 类对象的数组的对应位置中(注意新构造的对象的行数和列数)，函数结束时返回该新对象。

9. 矩阵转置 MAT<T> MAT<T>::operator~()；

如果操作符操作对象为空，抛出异常；构造一个新的对象，该对象的行数和列数与操作符操作对象相反，然后通过 for 循环对新对象的数组成员进行逐一赋值。

10. 矩阵深拷贝赋值 MAT<T>& MAT<T>::operator=(const MAT& a)；

如果参与赋值运算的两个矩阵相同，直接结束函数，如果被赋值的矩阵本身有不为

空的数组，先使用 delete 运算符释放内存。使用 new 运算符给被赋值对象的数组指针 e 分配与 a 中数组等大的空间，将被赋值对象行数列数分别赋值为 a.r 和 a.c，将 a 中数组内容赋值给被赋值对象数组中对应位置。

11. 矩阵移动赋值 MAT<T>& MAT<T>::operator=(MAT&& a)；
    如果参与赋值运算的两个矩阵相同，直接结束函数，如果被赋值的矩阵本身有不为空的数组，先使用 delete 运算符释放内存。将被赋值对象的数组指针、行数和列数分别赋值为 a 中数组的首地址、a.r、a.c；然后将 a 中数组指针、r、c 均赋值为 0 表示已被移动。

12. 矩阵运算“+=”MAT<T>& MAT<T>::operator+=(const MAT& a)；
    操作与矩阵加法 MAT<T> MAT<T>::operator+(const MAT& a)基本相同，区别在于不需要构造新的对象存放结果，而是将加法的结果赋值给运算符操作的对象，函数结尾返回对象自己。

13. 矩阵运算 “-=”MAT<T>& MAT<T>::operator-=(const MAT& a)；
    操作与矩阵减法 MAT<T> MAT<T>::operator-(const MAT& a)基本相同，区别在于不需要构造新的对象存放结果，而是将减法的结果赋值给运算符操作的对象，函数结尾返回对象自己。

14. 矩阵运算 “*=”MAT<T>& MAT<T>::operator*=(const MAT& a)；
    操作与矩阵乘法 MAT<T> MAT<T>::operator*(const MAT& a)基本相同，区别在于不需要构造新的对象存放结果，而是将乘法的结果赋值给运算符操作的对象，函数结尾返回对象自己。

15. 矩阵内容输出 char* MAT<T>::print(char* s)；
    利用字符串流对象 ss，将矩阵数组的内容以一定格式输入其中，再使用方法 ss.str()将内容复制到 string 类对象 str 中，最后通过方法 str.c_str()将内容赋值给结果 s，函数结尾返回 s。这种办法的优点在于可以规范输出不同数据类型的矩阵内容，而避免了重复写大段代码来对应不同数据类型的矩阵。

3 软件开发与测试

3.1 软件开发

硬件环境：

- 处理器：AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- 机带 RAM：16.0GB(15.4GB 可用)
- 系统类型：64 位操作系统，基于 x64 的处理器

开发环境：MSVC 2019 C++11，编译模式为 X86。 部分代码在 gcc， gdb 环境调试完成。

3.2 软件测试

使用实验提供的测试库函数得到结果如图 3.1 所示：

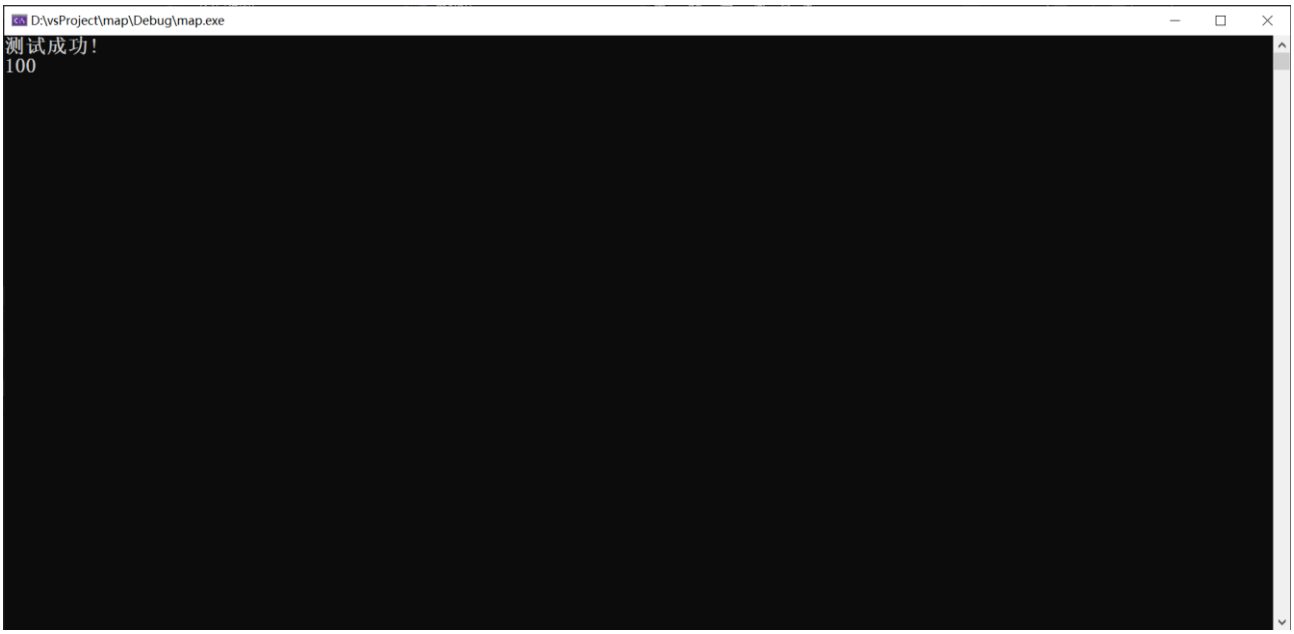

图 3.1 测试结果 1

4 特点与不足

4.1 技术特点

使用一维数组模拟二维数组的操作，同时使用 C++中的模板，可以同时对整形矩阵和长整型矩阵进行相关的操作。

4.2 不足和改进的建议

本次实验所涉及的功能较少，可以增加一些矩阵常见的操作，如求矩阵的闭包，而这也正是实验五所需要用到的矩阵的功能，另外也可以仿照 matlab 中的矩阵，去实现一个广义的矩阵，功能更加强大。

5 过程和体会

5.1 遇到的主要问题和解决方法

1. 第四次实验用到了 C++模板的相关内容，由于之前没有接触过，导致在实现的时候频繁出错，在最开始的静态语法检查的时候就会由于模板相关的语法规范而造成各种各样的问题，最终经过自己查阅资料，在老师和同学们的帮助下成功解决了无法运行的问题。

2. 在实现矩阵转置运算的时候，由于本身我对这个运算符的重载没有很明确的理解，导致最开始一直写错，后来甚至直接把调用转置运算符的矩阵值给改了，导致误以为自己*=运算写的有问题，最终在同学的帮助下找出了这个问题并修改。

3. 在进行本次实验的时候，我对内存管理是没有很明确的意识的，导致在很多地方直接 new 了一个新矩阵，但是在后来并没有将它 delete 掉，导致发生了很严重的内存泄漏，后来在老师的指导下知道了这个问题，并将其解决。

5.2 课程设计的体会

通过本次实验，我对 C++中的模板方面的知识有了更进一步的了解，知道了在使用模板的时候的一些注意事项，知道了含有模板的程序无法运行所存在的一些可能的原因，同时我对内存管理有了更加明确的认知，另外，本次实验实现了矩阵的部分运算，我也通过本次实验对矩阵有关的知识进行了复习。

6 源码和说明

6.1 文件清单及其功能说明

在提交的文件中，mat.h 文件存放矩阵类的声明和其方法的 1 定义，main.cpp 文件存放测试代码。

6.2 用户使用说明书

使用 vs2019 打开，在本地运行。

6.3 源代码

mat.h:

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iomanip>
#include <exception>
#include <typeinfo>
#include <string.h>
#include<stdio.h>
using namespace std;
template <typename T>
class MAT {
    T* const e;                         //指向所有整型矩阵元素的指针
    const int r, c;                     //矩阵的行 r 和列 c 大小
public:
    MAT(int r, int c);                  //矩阵定义
    MAT(const MAT& a);                  //深拷贝构造
    MAT(MAT&& a)noexcept;               //移动构造
    virtual ~MAT()noexcept;
    virtual T* const operator[ ](int r);//取矩阵 r 行的第一个元素地址，r 越界抛异常
    virtual MAT operator+(const MAT& a)const;      //矩阵加法，不能加抛异常
    virtual MAT operator-(const MAT& a)const; //矩阵减法，不能减抛异常
    virtual MAT operator*(const MAT& a)const;      //矩阵乘法，不能乘抛异常
```

```
    virtual MAT operator~()const;                    //矩阵转置
    virtual MAT& operator=(const MAT& a);          //深拷贝赋值运算
    virtual MAT& operator=(MAT&& a)noexcept;    //移动赋值运算
    virtual MAT& operator+=(const MAT& a);         //“+=”运算
    virtual MAT& operator-=(const MAT& a);         //“-=”运算
    virtual MAT& operator*=(const MAT& a);         //“*=”运算
//print 输出至 s 并返回 s：列用空格隔开，行用回车结束
    virtual char* print(char* s)const noexcept;
};
template<class Type>
MAT<Type>::MAT(int r, int c) :c(c), r(r), e(new Type[r * c])
{}
template <class Type>
MAT<Type>::MAT(const MAT& a) : c(a.c), r(a.r), e(new Type[a.r * a.c])
{
    for (int i = 0; i < r * c; i++)
    {
        e[i] = a.e[i];
    }
}
template <class Type>
MAT<Type>::MAT(MAT&& a)noexcept :c(a.c), r(a.r), e(a.e)
{
    *(int**)&a.e = NULL;
    *(int*)&(a.r) = 0;
    *(int*)&(a.c) = 0;
}
template<class Type>
MAT<Type>::~MAT()noexcept
{
    if (e != NULL)
        delete e;
    *(int**)&e = NULL;
    *(int*)&r = 0;
    *(int*)&c = 0;
}
```

```
template<class Type>
Type* const MAT<Type>::operator[](int r)
{
    if (r < 0 || r >= this->r)
    {
        throw("error!");
        return &e[0];
    }
    return e + r * c;
}
template<class Type>
MAT<Type> MAT<Type>::operator+(const MAT<Type>& a)const
{
    if (r != a.r || c != a.c)
    {
        throw("Can not add two matrices!");
        return *this;
    }
    MAT ans(a.r, a.c);
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            ans[i][j] = ((MAT)(*this))[i][j] + ((MAT)a)[i][j];
    return ans;
}
template<class Type>MAT<Type>
    MAT<Type>::operator-(const MAT<Type>& a)const
    {
        MAT ans(a.r, a.c);
        for (int i = 0; i < r * c; i++)
        {
            ans.e[i] = e[i] - a.e[i];
        }
        return ans;
    }
    template<class Type>
    MAT<Type> MAT<Type>::operator*(const MAT<Type>& a)const
```

```
{
    if (c != a.r)
    {
        throw("Can not multiply two matrices!");
        return *this;
    }
    MAT ans(r, a.c);
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < a.c; j++)
        {
            int sum = 0;
            for (int k = 0; k < c; k++)
            {
                sum += e[k + i * c] * a.e[j + k * a.c];
            }
            ans.e[i * r + j] = sum;
        }
    }
    return ans;
}
template<class Type>
MAT<Type> MAT<Type>::operator~()const
{
    MAT m(c, r);
    for (int i = 0; i < c; i++)
    {
        for (int j = 0; j < r; j++)
        {
            int t = i * r + j;
            m.e[t] = e[t / r + (t % r) * c];
        }
    }
    return m;
}
template<class Type>
```

```
MAT<Type>& MAT<Type>::operator=(const MAT<Type>& a)
{
    if (this == &a)
    {
        return *this;
    }
    if (e != NULL)
    {
        delete(e);
    }
    *(Type**)&e = new Type[a.r * a.c];
    for (int i = 0; i < a.r * a.c; i++)
    {
        e[i] = a.e[i];
    }
    *(int*)&r = a.r;
    *(int*)&c = a.c;
    return *this;
}
template<class Type>
MAT<Type>& MAT<Type>::operator=(MAT<Type>&& a)noexcept
{
    if (this == &a)
    {
        return *this;
    }
    if (e != NULL)
    {
        delete(e);
    }
    *(Type**)&e = a.e;
    *(int*)&r = a.r;
    *(int*)&c = a.c;
    *(int**)&a.e = NULL;
    *(int*)&a.r = 0;
    *(int*)&a.c = 0;
```

```
    return *this;
}
template<class Type>
MAT<Type>& MAT<Type>::operator+=(const MAT<Type>& a)
{
    if (r != a.r || c != a.c)
    {
        throw("Can not add two matrices!");
        return *this;
    }
    for (int i = 0; i < r * c; i++)
    {
        e[i] += a.e[i];
    }
    return *this;
}
template<class Type>
MAT<Type>& MAT<Type>::operator-=(const MAT<Type>& a)
{
    if (r != a.r || c != a.c)
    {
        throw("Can not add two matrices!");
        return *this;
    }
    for (int i = 0; i < r * c; i++)
    {
        e[i] -= a.e[i];
    }
    return *this;
}
template<class Type>
MAT<Type>& MAT<Type>::operator*=(const MAT<Type>& a)
{
    if (c != a.r)
    {
        throw("Can not multiply two matrices!");
```

```
    }
    MAT<Type> temp(*this);
    temp = *this * a;
    for (int i = 0; i < r * c; i++)
        e[i] = temp.e[i];
    return *this;
}
template<class Type>
char* MAT<Type>::print(char* s)const noexcept
{
    char temp[2000];
    if (typeid(e[0]) == typeid(int))
    {
        for (int i = 0; i < r; i++)
        {
            for (int j = 0; j < c; j++)
            {
                if (j == c - 1)
                {
                    sprintf(temp, "%6d\n\0", (int)e[i * c + j]);
                }
                else
                {
                    sprintf(temp, "%6d \0", (int)e[i * c + j]);
                }
                strcat(s, temp);
            }
        }
    }
    else if (typeid(e[0]) == typeid(long long))
    {
        for (int i = 0; i < r; i++)
        {
            for (int j = 0; j < c; j++)
            {
                if (j == c - 1)
```

```
                {
                    sprintf(temp, "%6lld\n\0", (long long)e[i * c + j]);
                }
                else
                {
                    sprintf(temp, "%6lld \0", (long long)e[i * c + j]);
                }
                strcat(s, temp);
            }
        }
    }
    return s;
}
template MAT<int>;
template MAT<long long>;
```

main.cpp:

```
#include"mat.h"
extern const char* TestMAT(int& e);
int main()                          //请扩展main()测试其他运算
{
    int e;
    const char* s;
    s = TestMAT(e);
    printf("%s\n%d", s, e);
}
```